Télécom Paris

IMA 206
Implémentation d'un papier de recherche
Rapport

Self-supervised learning - SimClr

Élèves :
Maël LE GUILLOUZIC
Arthur NUVOLONI
Noa ANDRE
Mohamed MOHAMED EL BECHIR

Enseignant :
Pietro GORI

26 juin 2025

Table des matières

1 Introduction

La production de données annotées et de qualité représente un frein important au développement de modèles performants en machine learning, en particulier en vision par ordinateur, où l'annotation est souvent coûteuse, longue et soumise à la variabilité humaine.

Pour répondre à cette problématique, les méthodes d'apprentissage auto-supervisé, notamment le contrastive learning, ont gagné en popularité ces dernières années. Elles permettent de limiter grandement les besoins en données annotées, et parfois, au prix d'un coût de calcul plus élevé, de dépasser les performances obtenues par des modèles entraînés en supervision classique.

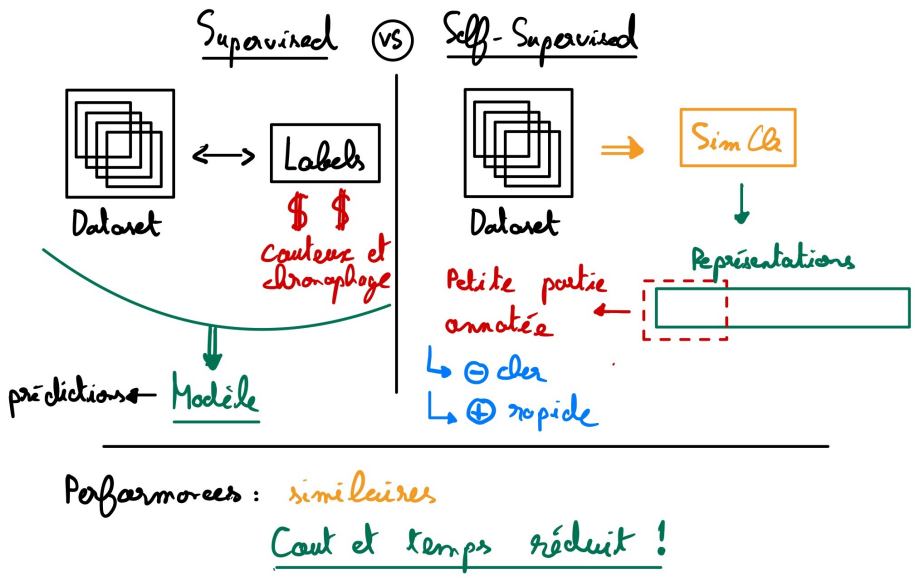

FIGURE 1 – Moins de données à annoter = coût et temps réduit !

Parmi ces approches, **SimCLR** (Simple Framework for Contrastive Learning of Visual Representations) s'est imposé comme une méthode de référence grâce à sa simplicité conceptuelle et son efficacité.

Pour ce projet, nous travaillons sur le jeu de données PathMNIST, issu du dataset plus large MedMNIST et qui propose des jeux de données standardisés et pré-traités. Notre jeu de données est ainsi constitué d'un peu plus de 107 000 images divisées en train, validation et test sets. Les images étant disponibles en différentes résolutions, nous avons débuté avec des images 28x28 pour implémenter un pipeline fonctionnel. Puis, tous les tests détaillés ci dessous, ont étés réalisés avec des images 128x128.

Notre projet s'est alors articulé autour de trois objectifs principaux :

1. Implémenter et à optimiser le framework SimCLR, et l'optimiser afin de pouvoir réaliser nos tests en un temps raisonable

2. Evaluer l'impact des différents hyperparamètres, et d'architectures différentes, sur les performances de notre modèle.
3. Comparer cette méthode avec d'autres. Nous avons ainsi implémenté une méthode plus récente : BarlowTwins, et chercher à positionner ces deux méthodes par rapport à des modèles de fondation tels que DINO v2 et CLIP.

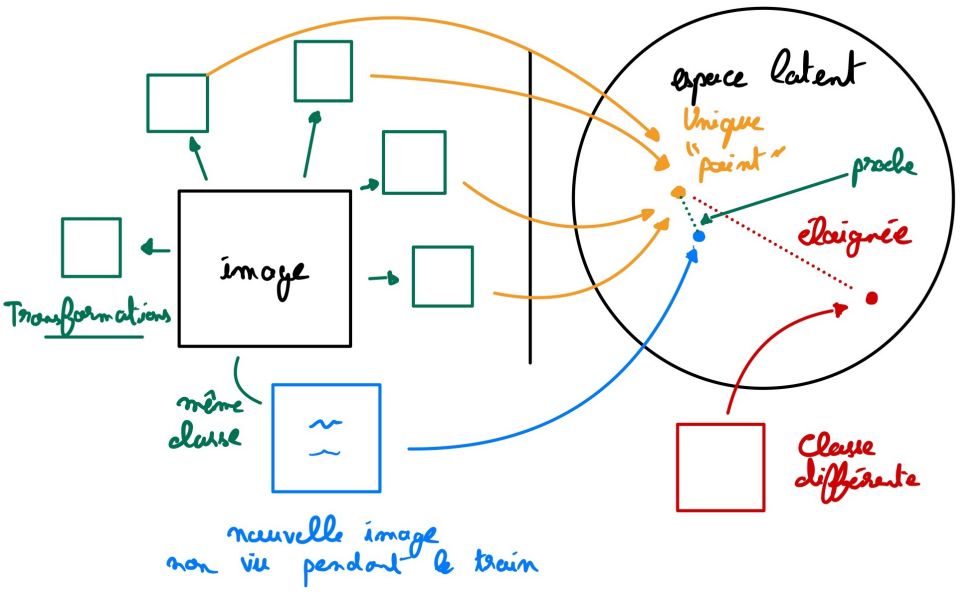

FIGURE 2 – Schéma de principe du Self Supervised Learning

2 SimCLR

2.1 Principe et architecture

SimCLR est une méthode publié en 2020 par une équipe de DeepMind, et propose une approche d'apprentissage auto-supervisé plutôt simple conceptuellement. Néanmoins cette méthode a permis à l'époque d'atteindre le même niveau de performances que des méthodes supervisées, avec seulement 20% de données annotées. Son objectif est d'apprendre des représentations visuelles riches, en se basant uniquement sur des transformations d'images et une fonction de perte contrastive.

Comme on peut le voir sur le schéma, SimCLR s'articule autour de quatre étapes principales :

- **Augmentation**

  Pour chaque image x d'un batch, nous générons deux images transformées différentes $\tilde{x}_i$ et $\tilde{x}_j$. Le but est altérer significativement l'apparence de l'image tout en préservant son contenu sémantique. Pour cela, de nombreuses transformations sont possibles, nous utilisons principalement :

  — Crop de taille et positionnement aléatoire dans l'image (limité entre 30% et 70% de la taille originale)

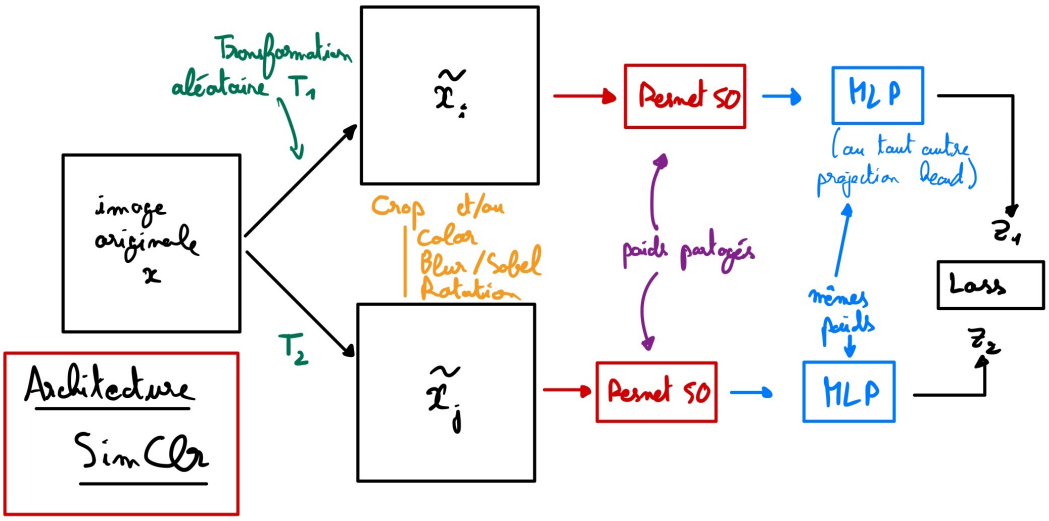

FIGURE 3 – Schéma de l'architecture SimClr

— Modification des couleurs (color jittering)
— L'application soit d'un flou gaussien, ou d'un filtre de Sobel
— Rotation d'un angle aléatoire

- **Encodeur**

  L'encodeur $f(\cdot)$ est un réseau de neurones qui transforme les images augmentées en représentations vectorielles. Nous avons choisi d'utiliser un ResNet-50, qui produit pour image un vecteur de sortie de taille

$$\mathbf{h} = f(\tilde{x}) \in \mathbb{R}^{2048}$$

  Le choix se pose alors de le prendre préentrainé, Resnet pouvant l'être sur ImageNet. Comme nous l'avons constaté lors de nos expérimentations ci dessous, bien que ImageNet et PathMNIST n'aient a priori rien à voir, le choisir *pretrained* permet d'accélérer notablement la convergence.

FIGURE 4 – Illustration des Datasets ImageNet et PathMnist

- **Projection Head**

  La projection Head $g(\cdot)$ est un petit réseau (MLP par exemple) qui vient se positionner entre l'encodeur et la loss, et projette les représentations $\mathbf{h}$ dans un espace de dimension plus réduite.

  $$\mathbf{z} = g(\mathbf{h}) \in \mathbb{R}^{128}$$

  C'est dans ce nouvel espace qu'on effectue la comparaison contrastive.

  Nous avons testé diverses structure pour cette projection head. Celle de la baseline consiste en un MLP avec une seule couche cachée, de taille 512 :

  $$g(\mathbf{h}) = W_2 \cdot \mathrm{ReLU}(W_1 \cdot \mathbf{h}) \tag{1}$$

  où $W_1 \in \mathbb{R}^{2048\times512}$ et $W_2 \in \mathbb{R}^{512\times128}$.

  Il est important de noter que cette tête de projection n'est utilisée **que pendant l'entraînement** ! Lors de l'évaluation, seul l'encodeur $f(\cdot)$ est conservé, et une nouvelle projection head est alors ré-entrainée.

- **Contrastive Loss**

  La contrastive Loss constitue le coeur du papier. Son fonctionnement est simple : pour un batch de N images, nous générons $2N$ images transformées. Pour chaque paire positive (i, j) provenant de la même image, la perte est définie comme :

  $$\mathcal{L}_{i,j} = -\log \frac{\exp(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{2}$$

où $\mathrm{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^T\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$ est la cosine similarity, τ est un paramètre de température qui contrôle la concentration de la distribution, et $\mathbb{1}_{[k\neq i]}$ est une fonction indicatrice égale à 1 si $k \neq i$ et 0 sinon.

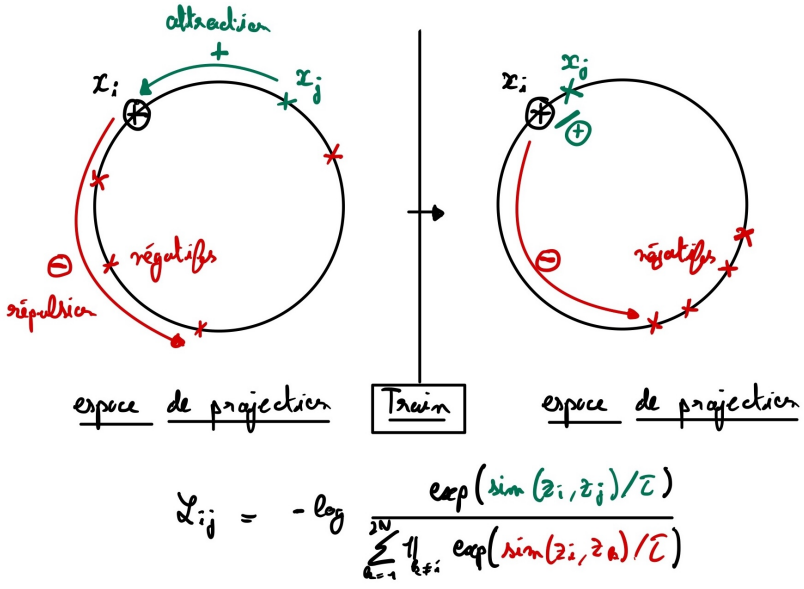

FIGURE 5 – Contrastive Loss avant et après l'entrainement

La perte finale est calculée comme étant la somme des losses sur toutes les paires du batch :

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} [\mathcal{L}_{2i-1,2i} + \mathcal{L}_{2i,2i-1}] \tag{3}$$

2.2 Implémentation Naïve

Notre implémentation initiale de SimCLR présentait deux goulots d'étranglement majeurs qui limitaient considérablement notre capacité à mener des expériences sérieuses. Nous avons donc développé deux optimisations cruciales qui ont significativement amélioré les performances et permis d'envisager des tests :

- **1. Vectorisation de la Loss**
  L'implémentation naïve de la loss utilisait une double boucle for sur les $2N$ vues du batch. En augmentant la taille du batch, le temps de calcul de cette dernière explose.
  Nous l'avons donc **vectorisé** en utilisant des opérations matricielles. Cela nous a permis de réduire le temps de calcul de la loss d'un facteur 20 pour des batchs de taille 256 !
- **2. Transformations sur GPU avec Kornia**
  Deuxième soucis de notre implémentation initiale : les augmentations de données étaient effectuées sur CPU via la bibliothèque torchvision, puis les images transformées étaient transférées sur GPU pour le calcul de la forward pass. Ces allers-retours CPU-GPU ralentissent considérablement l'entrainement, surtout lorsqu'on souhaite entrainer sur 100 000 images et 100 epochs.

  Nous avons donc implémenté les transformations directement sur GPU en utilisant la bibliothèque **Kornia**. Cela réduit aussi drastiquement les temps de calcul, particulièrement pour les batchs de grande taille et les images de résolution élevée (128×128).

Ces deux optimisations ont été cruciales pour nous permettre de mener les expériences décrites dans les sections suivantes, et en particulier l'analyse des hyperparamètres qui nécessite de nombreux entraînements complets.

Nous voici ainsi avec un pipeline d'entrainement complet. Néanmoins avant de détailler les résultats, il nous faut encore parler de Slurm et WandDB.

2.3 Déploiement sur cluster

Pour mener nos expériences de manière efficace et rigoureuse, nous avons mis en place une infrastructure d'expérimentation basée sur le système de gestion de jobs SLURM et l'outil de suivi d'expériences Weights & Biases (WandB).

2.3.1 Gestion des jobs avec SLURM

Le premier intêret de SLURM est de partager et d'optimiser un cluster de GPU entre divers utilisateurs et leurs différentes taches. En effet nos entrainement étaient prévus pour durer entre quelques heures et quelques dizaines d'heures. Il était donc opportun de trouver une solution qui nous permette de laisser tourner le job en tache de fond d'une part, et ne pas trop géner les autres utilisateurs d'autre part. Cette approche nous a permis de :

- Réserver précisément les ressources nécessaires (1 GPU et 16Go de RAM dans un premier temps, puis 2 GPUs et 32 Go de RAM) pour chaque expérience
- Exécuter plusieurs expériences en parallèle avec différents hyperparamètres
- Isoler chaque expérience dans son propre environnement d'exécution
- Créer un dossier de sortie unique pour chaque job, facilitant l'analyse des résultats, que nous voyons ci dessous sur wandb.

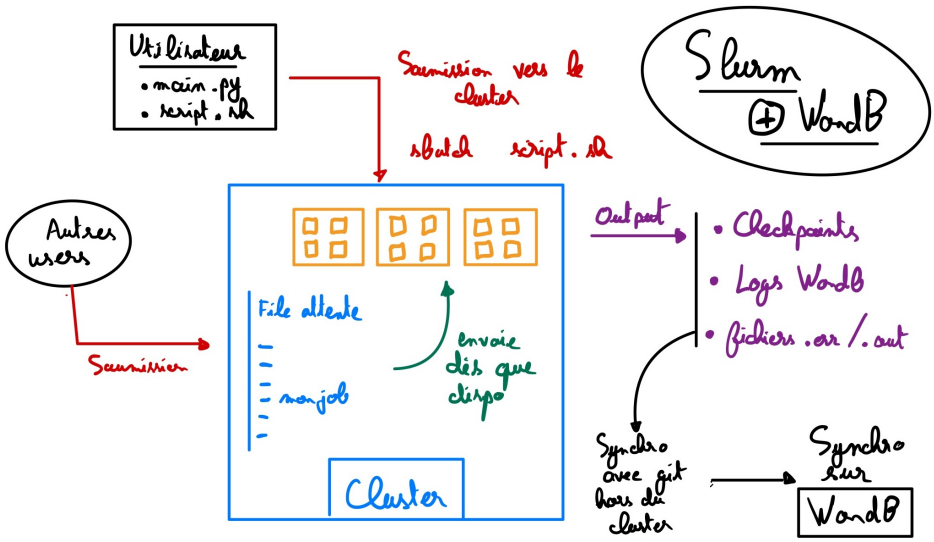

FIGURE 6 – Architecture Slurm + Wandb

2.3.2 Suivi des expériences avec WandB

Pouvoir observer facilement les résultat et surtout comparer les différentes runs était un point auquel nous tenions particulièrement. Le suivi et l'analyse de nos expériences fut ainsi réalisé avec Weights & Biases (WandB), une plateforme qui permet de :

- Suivre les métriques d'entraînement en temps réel (loss, temps d'exécution, et toute autre métrique désirée) si le GPU dispose d'un accès internet
- Comparer les performances de différentes configurations, et archiver les modèles et résultats voulus
- Toutes les runs sont visibles par tous les membres du groupe de travail

Néanmoins, le cluster Gpu-GW de l'Infres ne dispose pas d'accès internet (tout comme nombre de gros clusters, Jean Zay par exemple). Il a donc fallu adapter notre code pour utiliser le mode **"offline"** de WandB.

Dans ce mode, les données de chaque expérience sont d'abord stockées localement pendant l'exécution sur le cluster, puis synchronisées avec le service WandB une fois l'expérience terminée. Cette approche présente l'avantage que les ressources du cluster sont entièrement dédiées à l'entraînement, ce qui peut accélérer, mais de manière non substantielle. Plus utile, les données peuvent être validées et filtrées avant synchronisation.

En plus des métriques standard comme la perte et la précision, nous avons implémenté des métriques spécifiques à l'apprentissage auto-supervisé.
Par exemple, nous affichons sur wandb la similarité intra-classe et inter-classe dans l'espace des représentations. On essaie ainsi d'observer une mesure de la qualité des embeddings indépendamment du classifieur linéaire.

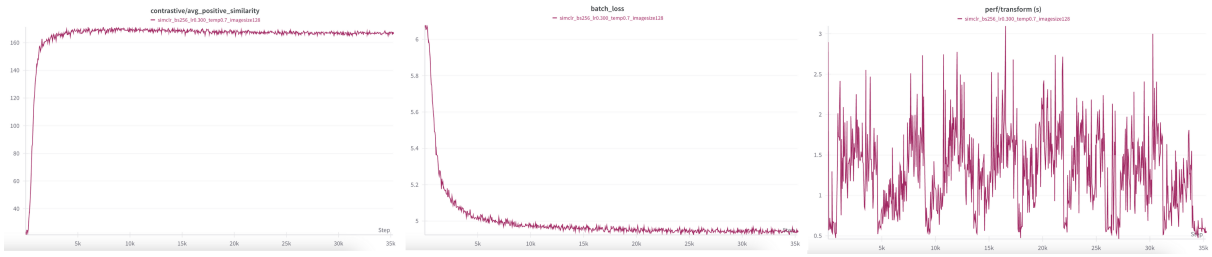

FIGURE 7 – Exemple de métriques affichables sur Wandb. Ici

3 Expérimentations et Résultats

Pour évaluer notre implémentation de SimCLR et comprendre l'impact des différents hyperparamètres sur les performances, nous avons conduit une série d'expériences. Cette section présente notre méthodologie d'évaluation et les résultats obtenus.

3.1 Méthodologie d'évaluation

Nous avons adopté la méthodologie standard du "linear probing" :

1. L'encodeur ResNet-50 est d'abord entraîné de manière auto-supervisée sur l'ensemble d'entraînement de PathMNIST, sans utiliser les étiquettes.
2. Une fois l'entraînement terminé, les poids de l'encodeur sont gelés et la tête de projection est retirée.
3. Un classifieur linéaire est ajouté au sommet de l'encodeur gelé et entraîné de manière supervisée pendant 50 époch, en utilisant cette fois les étiquettes des images.
4. La performance du classifieur sur l'ensemble de test est utilisée comme mesure de la qualité des représentations.

3.2 Impact de la taille du batch

La taille du batch est un paramètre critique pour l'apprentissage contrastif. En effet, elle détermine directement le nombre d'exemples négatifs disponibles pour chaque exemple positif. Dans la loss contrastive, pour chaque paire positive (i, j), les $2N - 2$ autres exemples du batch servent d'exemples négatifs.

Taille de batch	Précision (%)	Loss finale
64	79.8	0.96
128	73.3	1.59
256	84.0	0.75
512	76.4	1.26
1024	73.5	1.74

TABLE 1 – Impact de la taille du batch sur la précision et la loss

Nous avons testé les tailles de batch suivantes : **[64, 128, 256, 512, et 1024]**. L'objectif étant de comparer les tailles de batch et non pas de viser la performance la plus haute, ces runs ont été réalisées sur des images 28x28. Nos résultats montrent une relation non linéaire entre la taille du batch et les performances du modèle :

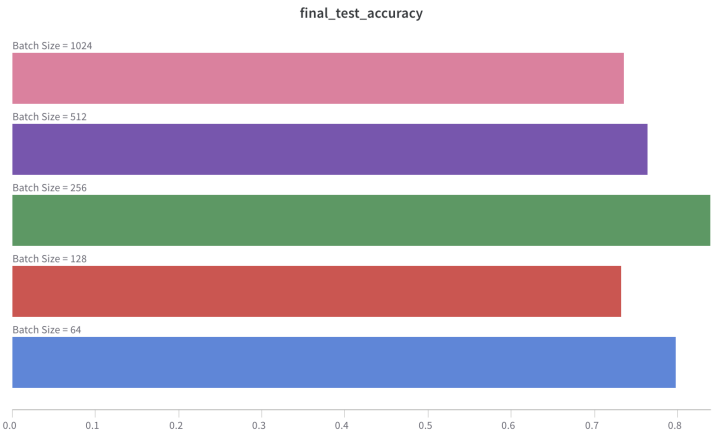

FIGURE 8 – Impact de la taille du batch sur l'accuracy

Nos observations principales sont :

- Nos performances ne croissent pas linéairement avec la taille du batch, contrairement à la littérature sur SimCLR
- La taille de batch optimale pour notre jeu de données est de 256, atteignant une précision de **84%**
- La loss des très grands batchs (1024) converge plus rapidement. Néanmoins comme nous avons réalisés tous les test avec le même nombre d'epochs, on obtient un écart entre loss d'entraînement et loss de test plus important. Ce problème introduit donc de l'overfitting, et diminue les performances des gros batch size.
- La taille 64, malgré son nombre limité d'exemples négatifs, performe étonnamment bien (81% d'accuracy)

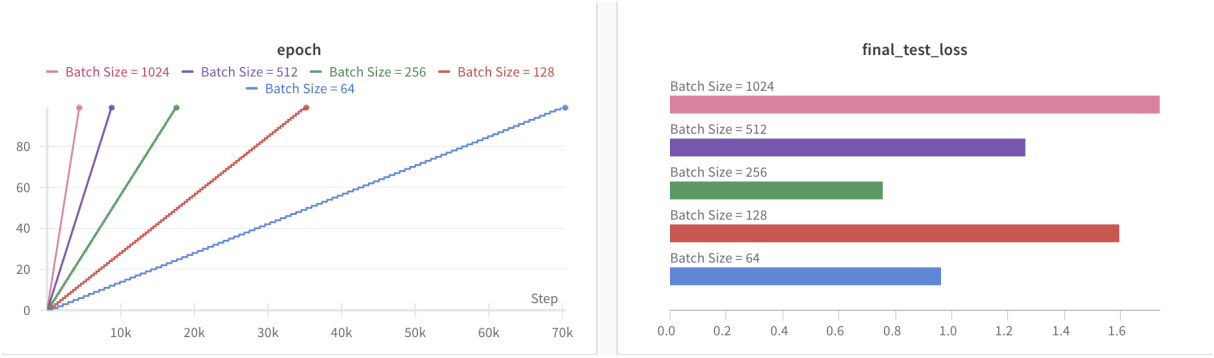

FIGURE 9 – Epochs et Test Loss

Ces résultats suggèrent que le batch size permet d'atteindre bien plus rapidement la convergence. Si on ne change ni le learning rate ni le nombre d'epochs, alors les performances diminuent de manière logique. Dans notre cas précis, une taille de batch intermédiaire (256) offre ainsi les meilleures performances.

3.3 Impact de la température

Le paramètre de température τ de la loss contrôle la *"dureté"* de la distribution de similarité. Une température plus basse accentue les différences entre les similitudes, poussant le modèle à se concentrer davantage sur les exemples difficiles.

Température	**Précision (%)**
0.7	0.792
1.0	0.638

TABLE 2 – Comparaison de deux paramètres de température.

Nous n'avons testé que deux températures différentes. Celle optimale est comme attendu la même que précisée dans le papier ($\tau = 0.7$).

3.4 Exploration des transformations d'augmentation

Les transformations appliquées aux images jouent un rôle fondamental dans l'apprentissage contrastif, si ce n'est le rôle principal. En effet, elles définissent implicitement ce que le modèle doit considérer comme invariant.
Nous avons systématiquement testé différentes combinaisons de transformations. La baseline comprend **Crop** + **Color** + **Blur**. Voici les résultats de nos expériences :
Nos observations principales sont :

- **Le recadrage aléatoire (crop) est essentiel** : L'absence de crop réduit significativement la précision (-6% par rapport à la baseline)
- **La simplicité est efficace** : La combinaison minimaliste "Crop + Blur" surpasse étonnamment les autres configurations avec 88.2% d'accuracy.
- **Les transformations de couleur ne fonctionnent pas** : étonnamment la combinaison "Crop + Blur" fonctionne aussi bien que la baseline "Crop + Color + Blur".

Configuration	Crop	Color	Blur	Rotation/Sobel	Accuracy (%)
Pas de Crop	✗	✓	✓	✗	81.9%
Crop + Blur	✓	✗	✓	✗	88.2%
Crop + Color + Sobel	✓	✓	✗	Sobel	81.9%
Crop+Color+Blur+Rotation	✓	✓	✓	Rotation	81.9%
Crop+Color+Blur (baseline)	✓	✓	✓	✗	88.2%

TABLE 3 – Impact des différentes combinaisons de transformations sur la précision finale.

C'est surprenant, car dans le papier SimCLr, le colorJitter est l'une des transformations les plus efficaces. Néanmoins le papier est rédigé sur ImageNet. Peut être que les transformations de couleurs que nous réalisons ont moins d'impact sur les images PathMNIST toutes roses.

- **Sobel est moins efficace que le flou gaussien** : La configuration avec Sobel au lieu du flou gaussien présente des performances inférieures. Ceci peut suggérer que les structures d'intéret résident plutot dans la disposition générale de l'image plutot que des contours précis.
- **La rotation n'améliore pas les performances** : La rotation diminue les performances obtenues.

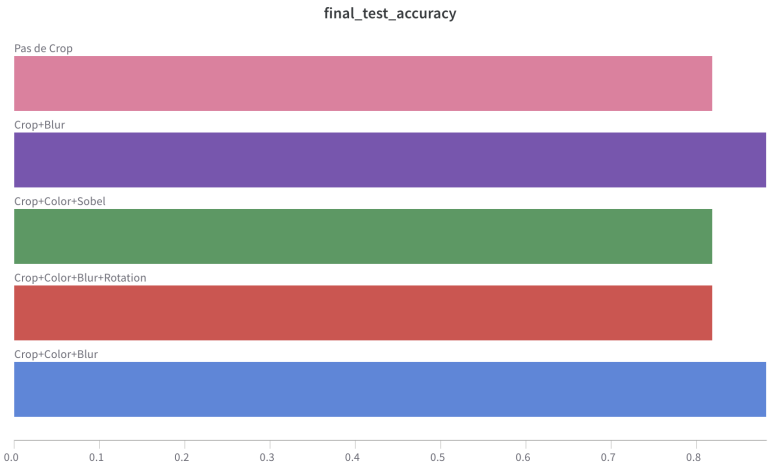

FIGURE 10 – Impact des différentes combinaisons de transformations sur la précision.

D'après ces résultats, on peut conclure qu'il est important d'adapter les transformations aux images sur lesquelles nous travaillons. Ainsi dans le cas de PathMNIST, l'apprentissage est meilleur en conservant un pipeline simple, et en se concentrant sur les structures générales plutot que sur les hautes fréquences spatiales.

3.5 Influence de l'architecture de la projection head

La projection head, bien que jetée après l'entraînement, joue aussi un rôle important dans la qualité des représentations apprises par l'encodeur.

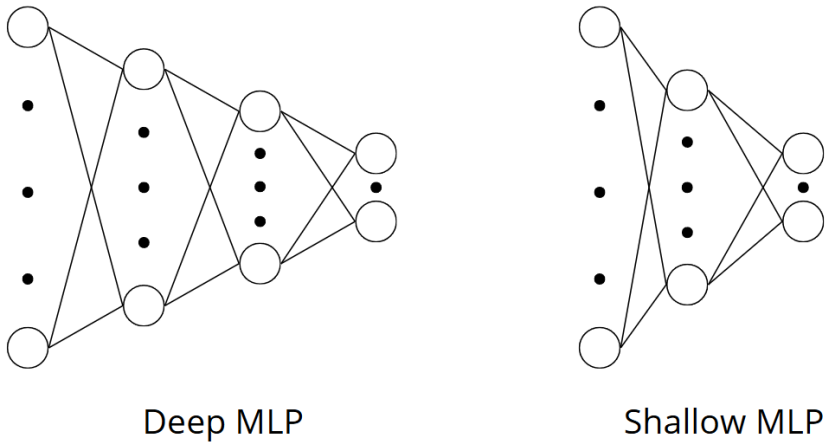

FIGURE 11 – Schéma des différentes projections head utilisées.

Nous avons expérimenté différentes architectures pour cette composante :

Architecture	**Dimensions**	**Params**	**Accuracy (%)**
MLP (Shallow) Linear	$2048 \to 512 \to 128$	1.11M	90.21
MLP (shallow)	$2048 \to 512 \to 128$	1.11M	92.38
MLP (deep)	$2048 \to 1024 \to 512 \to 128$	2.79M	92.43

TABLE 4 – Différentes architectures de projection head.

Les résultats montrent que :

- Une tête de projection linéaire est relativement efficace, même si les performances sont supérieures avec des activations non linéaires. Il est possible que l'encodeur produise des représentations suffisamment propices au contrastive learning, minimisant l'influence de la tête de projection.
- Une tête de projection non linéaire constituée d'une couche cachée est suffisamment performante pour le nombre de paramètres qu'elle possède. En effet, l'ajout d'une couche cachée ne modifie pas la précision de manière signifiante.

3.6 Pré-entraînement vs entraînement from scratch

Une autre question que nous nous sommes posés est de savoir si l'initialisation de l'encodeur avec des poids pré-entraînés sur ImageNet est bénéfique, malgré la différence considérable entre les datasets utilisés.

La précision finale atteint 91.12% avec pré-entraînement vs 92.38% sans pré-entraînement Ce résultat est **particulièrement intéressant et surprenant**. En effet, ou pouvait s'attendre à ce que le réseau prétrained performe mieux, puisque certaines caractéristiques visuelles de bas niveau, comme les canaux RGB, apprises sur des images naturelles restent pertinentes pour des images médicales. Mais ce n'est pas ce que nous observons.

Néanmoins en regardant la courbe de la loss, on remarque qu'elle atteint déja un plateau avant l'epoch 50. Ceci nous apprend ainsi deux choses. Premièrement, prendre

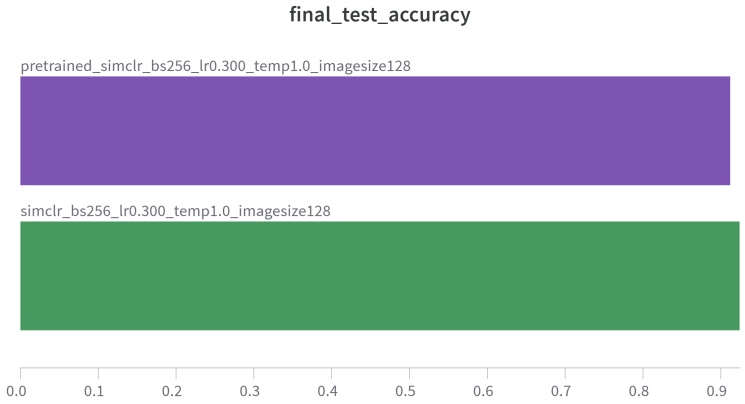

FIGURE 12 – Comparaison de la précision pour un encodeur pré-entraîné vs. initialisé aléatoirement.

un Resnet pretrained accélère la convergence. Mais en conservant un nombre d'epochs de 100, on overfit. C'est probablement la raison de nos performances moindres avec le pretrained.

3.7 Impact de la proportion de données avec label

L'intêret premier de l'apprentissage auto-supervisé est d'apprendre des représentations utiles avec un minimum de données étiquetées. Pour évaluer cet aspect crucial, nous avons testé l'impact de la proportion de données étiquetées utilisées lors de retrain du MLP (cut ratio).
Nous avons ainsi testé différentes proportions de données étiquetées : [10%, 20%, 40%, 100%]. A noter que pour la plupart des autres entrainements (baseline, et études des hyper-paramètres, ce ratio était de 20%).

Pourcentage de données étiquetées	**Précision (%)**	**Écart avec 20%**
10%	78.5	-0.7%
20%	79.2	0%
40%	91.2	+12%
100%	91.0	+11.8%

TABLE 5 – Impact du cut ratio sur l'accuracy, par rapport à la Baseline à 20%.

Nos observations principales sont les suivantes :

- **Augmentation du temps de retrain** : de manière logique, ré-entrainer avec plus de données implique un temps de retrain plus élevé.
- Avec seulement 10% des données étiquetées, nous atteignons **79.5%** de précision. C'est proche de la précision obtenue pour 20% de cut ratio, et donc signe que la phase principale d'apprentissage de SimClr est celle sans les labels. C'est plutot un bon indice sur le fonctionnement de notre modèle.

- L'utilisation de 20% des données permet d'atteindre 79.2%
- Le gain diminue significativement au-delà de 40% de cut ratio.

Ces résultats démontrent l'efficacité de SimCLR pour les scénarios où les données étiquetées sont limitées. C'est donc un très bon signe de fonctionnement de notre approche, puisqu'il signifie que le modèle pourra performer avec peu de données étiquetée.

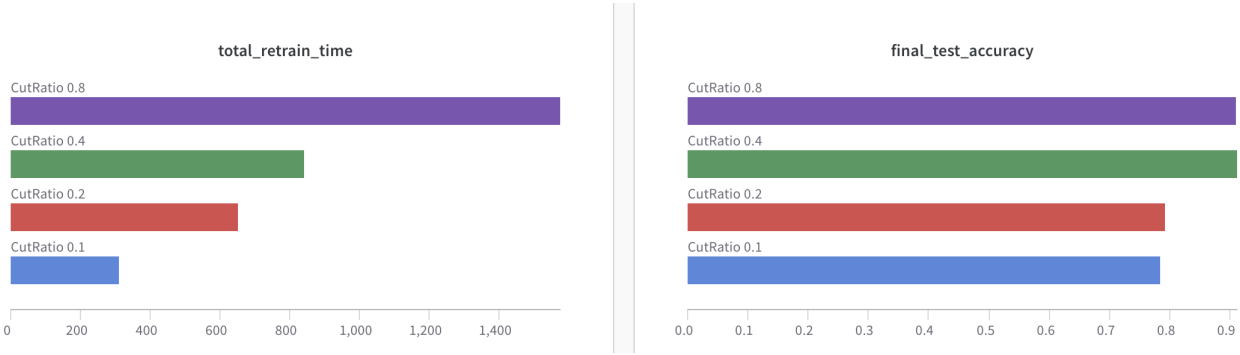

FIGURE 13 – Accuracy en fonction du Cut Ratio.

4 Barlow Twins

4.1 Présentation du papier

Barlow Twins est une méthode de self-supervised learning récente (2021) qui vise à réduire la redondance dans les représentations apprises. Elle introduit une nouvelle fonction de perte inspirée des travaux du neuroscientifique **H. Barlow**, notamment son principe de réduction de la redondance.

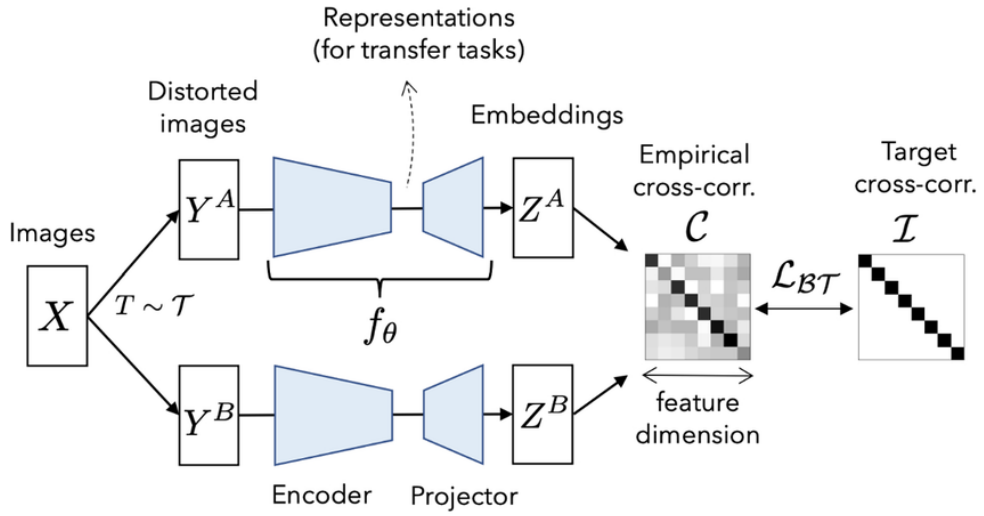

FIGURE 14 – Architecture de barlow Twins

L'architecture de Barlow Twins est globalement similaire à celle de *SimCLR*, ce qui a facilité son intégration dans notre pipeline. Le modèle repose sur un **ResNet-50** utilisé

comme encodeur, suivi d'un **MLP projecteur** composé de trois couches. Contrairement à d'autres méthodes, Barlow Twins utilise une dimension de sortie très élevée (souvent 4096 ou 8192), rendue nécessaire par la formulation particulière de sa fonction de perte, qui opère directement sur la matrice de corrélation des représentations.

4.2 Fonction de perte : Réduction de la redondance

Là où *Barlow Twins* se distingue particulièrement, c'est par l'introduction d'une nouvelle manière de calculer la fonction de perte. Contrairement à *SimCLR*, on ne compare pas les deux images transformées à l'ensemble des autres images du batch pour distinguer les paires positives et négatives, mais on utilise plutôt une matrice de corrélation croisée entre les deux vecteurs obtenus. Le but ici est de pousser le modèle à apprendre des représentations qui conservent l'information pertinente sur les données tout en étant invariantes aux distorsions appliquées aux échantillons.

$$\mathcal{L}_{\mathrm{BT}} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2$$

On y trouve un premier terme qui somme les éléments de la diagonal pour mesurer l'ivariance et s'assurer que les vecteurs sont similaires. Puis un deuxième terme sommants tous les autres éléments de la matrice, qui lui pénalise la redondance. Il est pondéré par un paramètre λ qui équilibre ces deux objectifs concurrents.

4.3 Robustesse à la taille du batch

Un des avantages notables de *Barlow Twins* est sa robustesse aux variations de la taille du batch. Contrairement à *SimCLR*, qui repose sur un grand nombre d'exemples négatifs issus du batch pour optimiser efficacement sa fonction de perte *InfoNCE*, *Barlow Twins* n'est que faiblement affecté par une réduction de la taille du batch. Ainsi, même avec des batchs de taille réduite, le modèle conserve de bonnes performances, ce qui le rend plus adapté à des environnements à ressources limitées.

4.4 Tests et résultats :

Nous avons testé ce modèle en repartant de la même base de code que pour notre implémentation de SimCLR, en modifiant uniquement ce qui était nécessaire pour adapter l'approche Barlow Twins. À partir de là, nous avons cherché à comprendre l'impact de trois paramètres principaux sur les performances du modèle :

- La taille du batch
- λ : le coefficient de pondération entre le terme d'invariance et le terme de redondance dans la fonction de perte
- La dimension de l'espace de projection

Pour avoir une base de référence stable, on a fixé certains paramètres pendant les tests : 70 epochs pour limiter le temps d'entraînement, un batch size de 128, et un $\lambda = 5 \cdot 10^{-3}$ comme recommandé dans le papier. À partir de cette configuration, on a fait varier un seul paramètre à la fois pour observer son impact.

On utilise un ResNet-50 non préentraîné, suivi d'un MLP avec une couche cachée de même dimension que la couche de sortie, et une normalisation par batch entre chaque couche.

L'évaluation est faite comme pour SimCLR : on réentraîne un MLP sur 20% des données pour mesurer la qualité des représentations. Voici les résultats obtenus :

Impact de la dimension de l'espace de projection :

Dimension de projection	Accuracy (%)
4096	84.8
8192	77.6

TABLE 6 – Impact de la dimension de l'espace de projection (avec $\lambda = 5 \cdot 10^{-3}$ et batch size = 128).

Contrairement à ce qui est suggéré dans l'article original, nous avons observé de meilleurs résultats avec une dimension de projection de 4096 plutôt que 8192.

Nous attribuons ce résultat au fait que nos données sont moins générales que celles utilisées dans le papier original, combiné à une taille de batch relativement réduite. Si la quantité d'information à capturer est limitée, un MLP trop puissant peut apprendre à déformer la géométrie de l'espace de représentation au lieu de favoriser une structure utile pour la classification en aval.

L'hyperparamètre Lambda :

λ	Accuracy (%)
1×10^{-4}	87.5
1×10^{-3}	87.0
5×10^{-3}	84.8
1×10^{-2}	86.9

TABLE 7 – Impact du coefficient λ (avec batch size = 128 et dimension de projection = 4096).

λ règle l'importance du terme qui force les dimensions à être décorrélées. On voit qu'une trop grande ou trop faible régularisation ne dégrade pas beaucoup la performance. La valeur recommandée ($5 \cdot 10^{-3}$) ne marche pas aussi bien sur notre jeu de donnée, mais des valeurs plus faibles comme 10^{-4} donnent légèrement mieux. Le modèle reste globalement stable face à ce paramètre.

Quand λ est très faible (par exemple 10^{-4}), la régularisation est minimale, ce qui pousse le modèle à privilégier l'alignement des vues augmentées. Ce comportement s'avère bénéfique pour notre tâche.

Quand λ est plus grand (10^{-2}), on force davantage la diversité des dimensions, ce qui peut aider à apprendre des représentations plus riches.

L'influence de la taille du batch :

Batch size	Accuracy (%)
64	91.6
128	87.0
256	90.8

TABLE 8 – Impact de la taille du batch (avec $\lambda = 1 \cdot 10^{-3}$ et dim de projection = 4096).

On a ici un résultat étonnant avec une accuracy record pour un batch de 64. On s'attendait à ce que le modèle soit robuste mais ne s'améliore pas non plus.

Cela s'explique sûrement par un effet de régularisation implicite : avec un petit batch, les gradients sont plus bruités, ce qui pousse le modèle à moins s'adapter aux détails spécifiques du batch et à mieux généraliser. À l'inverse, un batch de 128 donne les moins bons résultats, peut-être trop petit pour bien stabiliser l'entraînement, mais pas assez bruité pour régulariser efficacement.

5 Benchmark DINOv2 et CLIP

5.1 Extraction des représentations et classifieur linéaire

Ici nous comparons deux encodeurs visuels pré-entraînés à l'état de l'art — **DINOv2** et **CLIP** — pour l'extraction de représentations sur le jeu de données **PathMNIST**. Ces représentations sont ensuite évaluées à l'aide d'un classifieur linéaire (régression logistique), entraîné en régime de labels limités, avec des proportions de supervision variant de 1 % à 100 %. Pour chaque configuration, les hyperparamètres du classifieur sont ajustés de manière spécifique.

Les features DINOv2 sont extraites à l'aide du modèle `dinov2_vitl14` disponible via Torch Hub, tandis que celles de CLIP proviennent du modèle `ViT-B/16`, encapsulé dans une classe dédiée appelée `CLIPWrapper`.

5.2 Résultats

5.2.1 DINOv2 (ViT-L/14) et CLIP (ViT-B/16)

% Labels	Train Acc	Test Acc
1 %	0.9944	0.8731
20 %	0.9507	0.8976
50 %	0.9445	0.9024

TABLE 9 – Performances du classifieur linéaire sur les features DINOv2

Avec seulement 1 % des étiquettes, on observe que :

— **DINOv2** atteint 87,31 % de test Acc (train Acc 99,44 %), ce qui montre une capacité remarquable à généraliser malgré un très faible nombre de labels.

— **CLIP** se situe un peu en retrait à 86,07 % de test Acc (train Acc 97,22 %), suggérant un léger surapprentissage comparé à DINOv2 dans ce régime extrême.

% Labels	Train Acc	Test Acc
1 %	0.9722	0.8607
20 %	0.9548	0.8926
50 %	0.9502	0.8921

TABLE 10 – Performances du classifieur linéaire sur les features CLIP

6 Conclusion

Les résultats obtenus avec 20% de labels révèlent des différences intéressantes entre les méthodes.

Méthode	Backbone	Accuracy (%)
SimCLR	ResNet-50 (sans pré-entraînement)	92,38
Barlow Twins	ResNet-50 (sans pré-entraînement)	91,57
DINOv2	ViT-L/14 (pré-entraîné)	89,76
CLIP	ViT-B/16 (pré-entraîné)	89,26

TABLE 11 – Accuracy des différents modèles avec 20 % des labels annotés (PathMNIST)

Notre implémentation SimCLR obtient la meilleure performance avec une précision de 92,38 %. Cela montre qu'un bon entraînement contrastif peut surpasser des backbones déjà optimisés. L'usage d'augmentations combiné à une loss InfoNCE bien calibrée joue ici un rôle clé.

Barlow Twins, avec 91,57 %, arrive juste derrière. Sa loss de réduction de redondance semble offrir une stabilité supplémentaire, en particulier quand les faux appariements deviennent problématiques dans les approches purement contrastives.

DINOv2 (89,76 %) et **CLIP** (89,26 %) ferment la marche. Tous deux utilisent des ViT pré-entraînés sur de vastes corpus, mais leurs représentations générales s'adaptent un peu moins bien à la spécificité de PathMNIST.

En résumé, notre méthodes auto-supervisées implémentée reste plus performante car probablement entrainé spécifiquement sur nos images PathMNIST. Cela confirme que la majeure partie de l'apprentissage se déroule lors de la phase d'entrainement de l'encodeur.

Mot de fin

Ce projet nous a permis de découvrir l'apprentissage supervisé, ainsi que d'apprendre énormément sur l'utilisation de clusters avec Slurm et Wandb, merci pour cela.
PS : taille du rapport, 10 pages sans les figures, tableau et pages de garde

Références

[1] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). *A Simple Framework for Contrastive Learning of Visual Representations.*

[2] Zbontar, J., Jing, L., Misra, I., LeCun, Y., & Deny, S. (2021). *Barlow Twins : Self-Supervised Learning via Redundancy Reduction.*

[3] Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Masse, A., El-Nouby, A., Assran, M., Ballas, N., Allemand, L., Duval, A., Josse, A., Rodriguez, C., Picard, D., Vachet, G., Vurgaftman, A., Bojanowski, P. & Joulin, A. (2023). *DINOv2 : Learning Robust Visual Features without Supervision.*